TP3 : VBO, Maillages et indexation

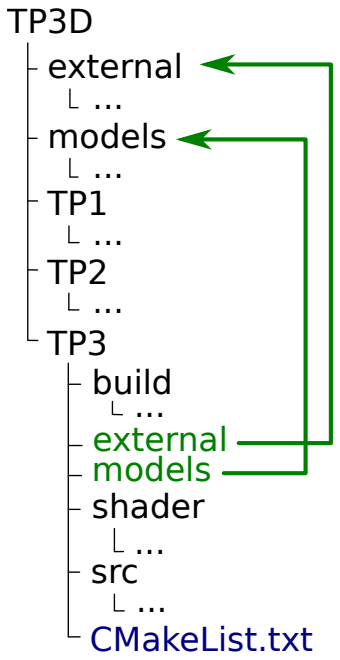
Introduction

L'objectif du TP est de s'initier à la génération et à l'importation de modèle 3D.

La base de code qui vous est fournie reprend les éléments de la dernière séance : création de fenêtre GLFW et initialisation de contexte OpenGL, affichage de géométrie, et utilisation des matrices de transformation pour manipuler la scène. Pour la récupérer, allez à l'adresse suivante :
https://team.inria.fr/imagine/modelisation-synthese-dimage-ricm4-polytech-2015-2016/.

Pour l'utiliser, faites comme la dernière fois :

- extrayez l'archive du TP dans le dossier de votre répertoire personnel dédié aux TP de 3D (`unzip TP3.zip -d ~/TP3D/`)
- extrayez l'archive du dossier contenant les modèles 3D au même endroit (`unzip models.zip -d ~/TP3D/`)
- accédez au dossier du TP (`cd ~/TP3D/TP3/`)
- créez un lien symbolique vers le dossier `external` (`ln -rs ../external/`)
- créez un lien symbolique vers le dossier `models` (`ln -rs ../models/`)
- créez un dossier pour la compilation (`mkdir build`)
- accédez à ce dossier (`cd build`)
- lancez `cmake` (`cmake ..`)
- lancez la compilation (`make`)
- exécutez (`./polytech_ricm4_tp3`)



Voici la structure que devrait avoir votre arborescence de fichiers.

Buffers

Pour rappel, un buffer ("tampon" en français) est une zone en mémoire sur la carte graphique qui contient des données utile pour l'affichage de l'image. Pour l'instant, nous avons seulement utilisé un buffer pour stocker les positions des sommets de nos objets. Nous allons maintenant voir comment créer des buffers pour gérer d'autres types de données : couleurs, normales, indices.

Comme pour le tableau de position, nous allons utiliser les `std::vector` pour représenter nos données sur le CPU. Cette structure a l'avantage de stocker l'information de manière contigue en mémoire. Pour ajouter un élément à un `std::vector`, utilisez la fonction `push_back()`. Pour connaître le nombre d'éléments déjà stockés, utilisez `size()`.

Créez un tableau de stockant une couleur par sommet du cube, que vous nommerez `colors`.

Ensuite, il faut créer un buffer à partir de `colors`. Pour ceci, utilisez les commandes suivantes :

```
GLuint colorBufferID;

// creation d'un nouveau buffer
glGenBuffers(1, &colorBufferID);

// definition de celui-ci comme le buffer courant
glBindBuffer(GL_ARRAY_BUFFER, colorBufferID);
```

```
// recopie des donnees de "colors" dans le buffer courant
glBufferData(GL_ARRAY_BUFFER, colors.size() * sizeof(vec3), color, GL_STATIC_DRAW);
```

Ensuite, il faut modifier le vertex shader pour que celui-ci réceptionne ce nouvel attribut :

```
#version 150

// Donnees d'entree
in vec3 in_position;
in vec4 in_color;

// Donnees de sortie
out vec4 vert_color;

// Parametres
uniform mat4 ModelMatrix;
uniform mat4 ViewMatrix;
uniform mat4 ProjectionMatrix;

// Fonction appellee pour chaque sommet
void main()
{
  // Affectation de la position du sommet
  gl_Position = ProjectionMatrix *ViewMatrix * ModelMatrix * vec4(in_position, 1.0);

  // Affectation de la couleur du sommet
  vert_color = in_color;
}
```

Ainsi que le fragment shader, pour que celui-ci affecte la valeur de couleur des sommets au fragment :

```
// Version d'OpenGL
#version 150

// Donnees d'entree
in vec4 vert_color;

// Donnees de sortie
out vec4 frag_color;

// Fonction appellee pour chaque fragment
void main()
{
  // Affectation de la couleur du fragment
  frag_color = vec4(vert_color.rgb ,1.0);
}
```

Enfin, il reste à récupérer l'identifiant de la variable `in_color` du vertex shader dans l'initialisation du main :

```
GLuint colorPositionID = glGetAttribLocation(programID, "in_color");
```

Avant de l'utiliser dans la boucle de dessin :

```
// Activation de l'attribut colorPositionID
glEnableVertexAttribArray(colorPositionID);

// Definition de colorBufferID comme le buffer courant
glBindBuffer(GL_ARRAY_BUFFER, colorBufferID);
```

```
// On indique comment lire les donnees
glVertexAttribPointer(
  colorPositionID,
  4,
  GL_FLOAT,
  GL_FALSE,
  0,
  (void*)0
);
```

Sans oublier de désactiver l'attribut à la fin du dessin :

```
glDisableVertexAttribArray(colorPositionID);
```

Et de détruire le buffer à la fin du programme :

```
glDeleteBuffers(1, &colorBufferID);
```

Buffer d'indices

Nous allons maintenant créer un buffer d'indice. L'idée est la suivante : au lieu de transmettre un sommet à chaque fois qu'on en a besoin, on va transmettre une seul fois tout les sommets, puis y faire référence grâce à des indices.

De même que pour les couleurs, créez un tableau contenant les indices des sommets :

```
vector<uint> indices;
indices.push_back(0);
indices.push_back(1);
indices.push_back(2);
indices.push_back(3);
indices.push_back(4);
indices.push_back(5);
```

Pour le reste, faites comme pour les couleurs et les positions, en remplaçant `GL_ARRAY_BUFFER` par `GL_ELEMENT_ARRAY_BUFFER`.

Indexing

Modifiez la définition des buffers précédents pour n'utilisez que 8 sommets (à la place de 36), en utilisant des indices de sommets.

```
OFF

#vertex_count face_count  edge_count
3     1     0

#One line for each vertex
0.0   0.5   0.0
0.5   -0.5  0.0
-0.5  -0.5  0.0

#One line for each polygonal face
#vertex_number vertex_indices
3 0 1 2
```

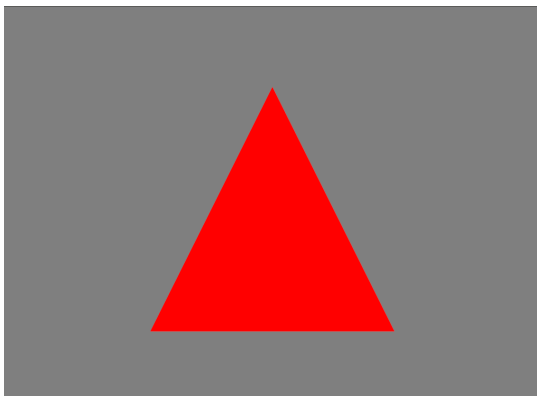


Figure 1: Exemple de fichier OFF pour un triangle

Maillage

Nous allons maintenant nous intéresser à un mode de représentation des objets en 3D : les maillages. Un maillage est une surface discrétisée sous la forme de polygones. Nous nous en servons en informatique graphique pour représenter les objets.

Un maillage est la donnée de :

- un ensemble de points, ils représentent les sommets des polygones formant notre maillage
- un ensemble d'indices, ils représentent l'appariement des sommets pour former les polygones

Pour stocker un maillage dans un fichier, il existe différents formats. Le plus simple d'entre eux est le format OFF. Pour plus de détail, voir figure 1.

Les fichiers "Mesh.h" et "Mesh.cpp" implémentent une classe permettant de représenter un maillage : cette classe contient des attributs pour les tableaux de positions, de normales, de couleurs, et d'indices. Par ailleurs, cette classe propose de charger automatiquement ces informations à partir d'un fichier OFF.

Pour l'utiliser, faites comme ceci :

```
Mesh m("../models/armadillo.off");   // chargement du maillage
m.vertices.data();                   // acces au tableau de positions
m.normals.data();                    // acces au tableau de normales
m.faces.data();                      // acces au tableau d'indices
```

Travail à faire : chargez un maillage du répertoire "models/" et affichez le dans la fenêtre.